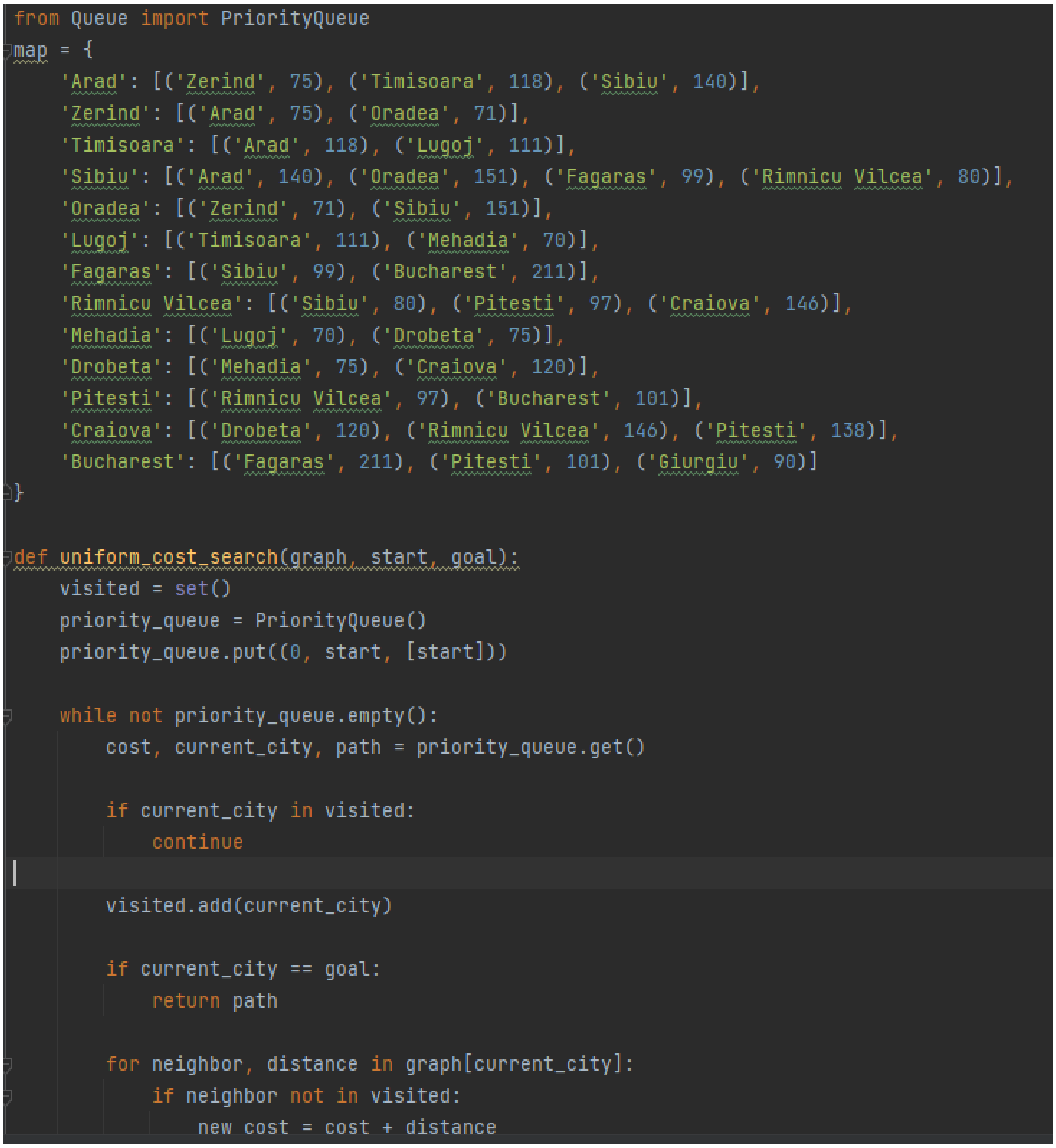
# ARTIFICIAL INTELLIGENCE (CSC 462)

## LAB # 6

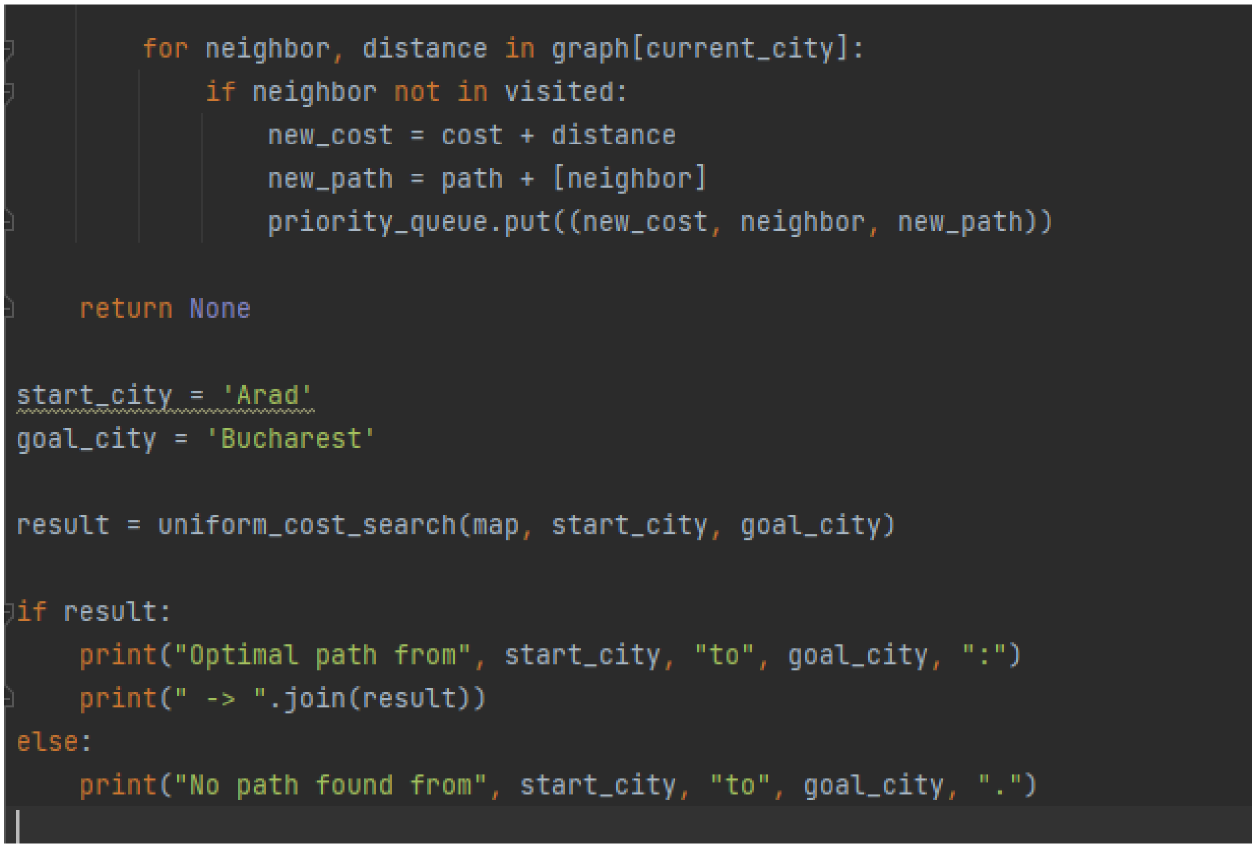
| | |
|---|---|
| **NAME:** | MUAAZ BIN MUKHTAR |
| **REG NO:** | FA21-BSE-045 |
| **CLASS & SECTION:** | BSSE-5A |
| **SUBMITTED TO:** | SIR WAQAS ALI |
| **DATE SUBMITTED:** | 16-10-2023 |

**Department of Computer Science**

**Imagine going from Arad to Bucharest in the following map. Your goal is to minimize the distance mentioned in the map during your travel. Implement a uniform cost search to find the corresponding path**.

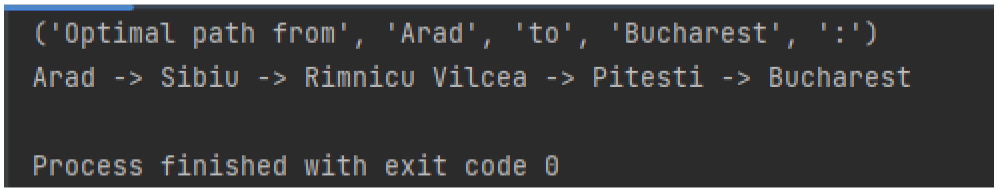


```python
from Queue import PriorityQueue
map = {
    'Arad': [('Zerind', 75), ('Timisoara', 118), ('Sibiu', 140)],
    'Zerind': [('Arad', 75), ('Oradea', 71)],
    'Timisoara': [('Arad', 118), ('Lugoj', 111)],
    'Sibiu': [('Arad', 140), ('Oradea', 151), ('Fagaras', 99), ('Rimnicu Vilcea', 80)],
    'Oradea': [('Zerind', 71), ('Sibiu', 151)],
    'Lugoj': [('Timisoara', 111), ('Mehadia', 70)],
    'Fagaras': [('Sibiu', 99), ('Bucharest', 211)],
    'Rimnicu Vilcea': [('Sibiu', 80), ('Pitesti', 97), ('Craiova', 146)],
    'Mehadia': [('Lugoj', 70), ('Drobeta', 75)],
    'Drobeta': [('Mehadia', 75), ('Craiova', 120)],
    'Pitesti': [('Rimnicu Vilcea', 97), ('Bucharest', 101)],
    'Craiova': [('Drobeta', 120), ('Rimnicu Vilcea', 146), ('Pitesti', 138)],
    'Bucharest': [('Fagaras', 211), ('Pitesti', 101), ('Giurgiu', 90)]
}

def uniform_cost_search(graph, start, goal):
    visited = set()
    priority_queue = PriorityQueue()
    priority_queue.put((0, start, [start]))

    while not priority_queue.empty():
        cost, current_city, path = priority_queue.get()

        if current_city in visited:
            continue

        visited.add(current_city)

        if current_city == goal:
            return path

        for neighbor, distance in graph[current_city]:
            if neighbor not in visited:
                new_cost = cost + distance
```



```
        for neighbor, distance in graph[current_city]:
            if neighbor not in visited:
                new_cost = cost + distance
                new_path = path + [neighbor]
                priority_queue.put((new_cost, neighbor, new_path))

    return None

start_city = 'Arad'
goal_city = 'Bucharest'

result = uniform_cost_search(map, start_city, goal_city)

if result:
    print("Optimal path from", start_city, "to", goal_city, ":")
    print(" -> ".join(result))
else:
    print("No path found from", start_city, "to", goal_city, ".")
```

**Output:**



```
('Optimal path from', 'Arad', 'to', 'Bucharest', ':')
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest

Process finished with exit code 0
```